IS 611:Advanced Object Oriented Programming (12 Credits)

Course Objectives

1. To equip students with knowledge on object-oriented programming principles and techniques,

2. To understand the differences between structured and object oriented programming paradigms,

3. Demonstrate ways in which object-oriented programming, in this case Java, facilitates code reusability in developing large and complex software.

IS 611:Advanced Object Oriented Programming (12 Credits)

Learning Outcomes

Upon completion, students should be able to:

i. Demonstrate an understanding of the underlying principles and concepts of Object-Oriented Programming

ii. Apply the concepts of data encapsulation, inheritance, and polymorphism to large-scale software

iii. Design and develop object-oriented computer programs

iv. Design and develop programs with Graphical User Interfaces capabilities

v. Use an object-oriented language to develop complex programs with team-work in mind

Development Environment

To write java program, you'll need:

One example of development environment are:

- **The Java SE Development Kit (JDK)**
  - For Microsoft Windows, Solaris OS, and Linux:
    https://www.oracle.com/technetwork/java/javase/downloads/index.html
  - For Mac OS X: https://developer.apple.com/

- **The NetBeans IDE**
  - For all platforms: https://netbeans.org/downloads/index.html

- **Setting up the environment and getting started**

https://www.oracle.com/java/technologies/getstarted-setup-java-programming.html

Java Classes and Objects

- A class forms the basis for object-oriented programming in Java.

- Any concept you wish to implement in a Java program must be designed by a class.

Java Classes and Objects

Simple analogy to help you understand classes and their contents:

A car typically begins as engineering drawings, similar to the blueprints used to design a house. These engineering drawings includes:

- The design for an accelerator pedal to make the car go faster. The pedal "hides" the complex mechanisms that actually make the car go faster
- The brake pedal "hides" the mechanisms that slow the car
- The Clutch – to help change the gears
- The steering wheel "hides" the mechanisms that turn the car.

This enables people with little or no knowledge of how engines work to drive a car easily.

Java Classes and Objects

Unfortunately, you cannot drive the engineering drawings of a car. Before you can drive a car, the car must be built from the engineering drawings that describe it →Building an object of a class

A completed car will have an actual accelerator pedal, However, the car will not accelerate on its own, so the driver must press the accelerator pedal.
Performing a task in a program requires a method. The method describes the mechanisms that actually perform its tasks.

The method hides from its user the complex tasks that it performs, just as the accelerator pedal of a car hides from the driver the complex Mechanisms of making the car go faster.

Java Classes and Objects

A car also has many attributes, such as:

- its color,
- the number of doors,
- the amount of fuel/gas in its tanks,
- its current speed and
- its total miles driven (i.e. its odometer reading).

These attributes are represented as part of a car's design in its engineering diagrams. Every car maintains its own attributes. For example, each car knows how much fuel is in its own fuel tank, but not how much is in the tanks of other cars.

- Attributes are specified by the class's instance variables

Java Classes and Objects

- The most important thing to understand about a class is that it defines a new data type.

- Once defined, this new type can be used to create objects of that type.

- Thus, a class is a *template for an object, and an object is an instance of a class.*

- A class is a logical framework that defines the relationship between its members.

- Thus, an object has physical reality (That is, an object occupies space in memory.)

Java Classes and Objects

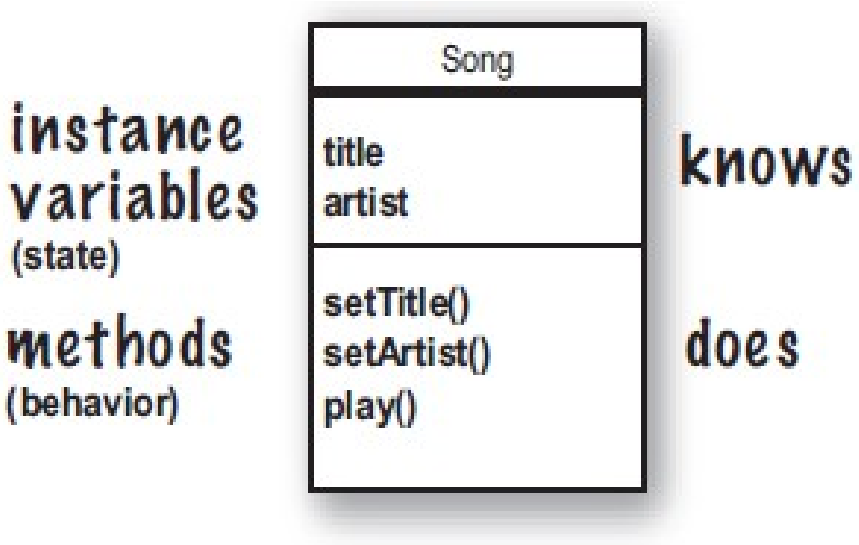
When you design a class, think about the objects that will be created from that class type. Think about:

- things the object knows
- things the object does

Things an object knows about itself are called instance variables

Things an object can do are called methods

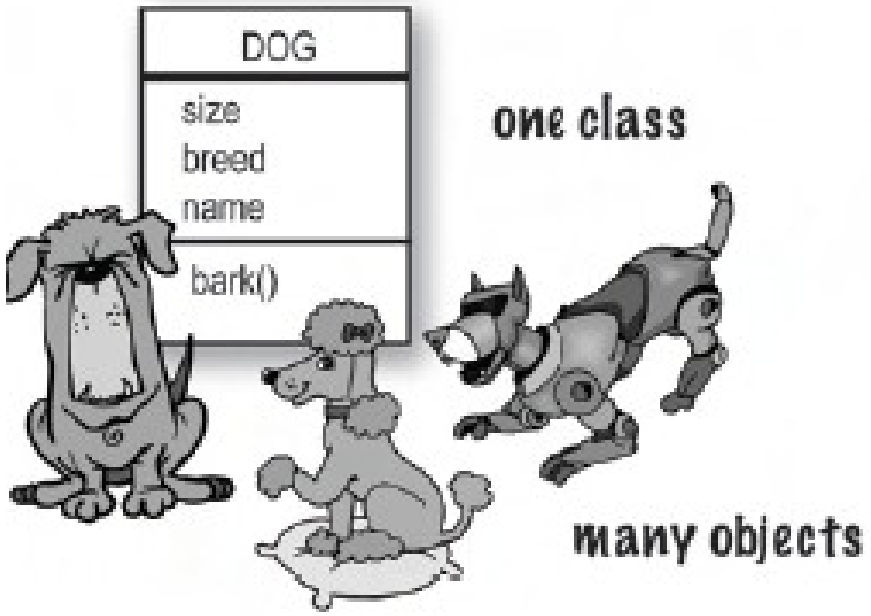
Class diagram:

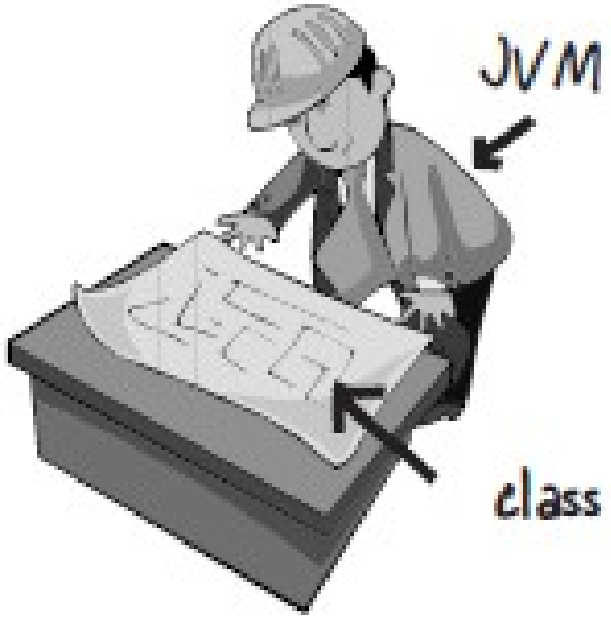


Java Classes and Objects

What's the difference between a class and an object?

A class is a blueprint for an object. It tells the virtual machine how to make an object of that particular type. Each object made from that class can have its own values for the instance variables fo that class.





Java Classes and Objects

1 Write your class

```
class Dog {

  int size;
  String breed;
  String name;

  void bark() {
    System.out.println("Ruff! Ruff!");
  }
}
```

instance variables

a method

Java Classes and Objects

GradeBook is an example class which contains one method (displayMessage()) that simply displays a welcome message:

Welcome to the Grade Book! when it is called.

```
public class GradeBook{
      public void displayMessage()
     {
     System.out.println("Welcome to the Grade Book!" );
      }
}
```

Here is another class called **Box** that defines three instance variables: **width, height, and depth.** Currently, Box does not contain any methods

```
class Box {
     double width;
     double height;
     double depth;
}
```

Class diagrams?

Java Classes and Objects

- Class GradeBook, class Box and class Dog are not java applications because they do not contain the main method. Therefore, if you execute e.g GradeBook.class (after compiling GradeBook.java) you will get an error message.

- To fix this problem, we must either declare a separate class that contains a main method or place a main method in class Dog, Box or GradeBook .

- But for larger programs in industries, a separate class containing method main is normally used to test a developed class.

- In this example, we create a new class BoxDemo, DogTestDrive and GradeBookTest to test the Box class, Dog class and GradeBook class:

Creating objects of the class & calling methods

3 In your tester, make an object and access the object's variables and methods

```
class DogTestDrive {
   public static void main (String[] args) {
     Dog d = new Dog();
     d.size = 40;
     d.bark();
   }
}
```

make a Dog object

use the dot operator (.) to set the size of the Dog

and to call its bark() method

dot operator

Creating objects of the class & calling methods

```
// This program includes a method inside the box class.
      class Box {
      double width;
      double height;
      double depth;
      // display volume of a box
      void volume() {
      System.out.print("Volume is ");
      System.out.println(width * height * depth);
      }
      }

      class BoxDemo{
                  public static void main(String [] args) {
                  Box mybox = new Box();
      // assign values to mybox´s instance variables
      mybox.width = 10;
      mybox.height = 20;
      mybox.depth = 15;
      // display volume of the box
      mybox.volume();
      }
      }
```

Methods that returns a value

The previous **volume( ) method calculates and displays value of volume once it is called.**

What if another part of the program wanted to know the volume of a box, but not display its value?

This can be sorted by implementing **volume( ) method by letting it compute the volume of the box and return the result** to the caller.

```
// Now, volume() returns the volume of a box.
    class Box {
            double width;  double height; double depth;
            // compute and return volume
              double volume() {
                return width * height * depth;
                                }
              }

    class BoxDemo2 {
            public static void main(String args[]) {
                 Box mybox1= new Box();
                double vol;
            // assign values to mybox's instance variables
            mybox1.width = 10; mybox1.height = 20; mybox1.depth = 15;
            // get volume of first box
              vol = mybox1.volume();
              System.out.println("Volume is " + vol);
            }
    }
```

Constructors

Typically, you can not call a method that belongs to another class until you create an object of that class.

This is done by the following line in class GradeBookTest:

```
GradeBook myGradeBook = new GradeBook();
```

Keyword new creates a new object of the class specified to the right of the keyword (i.e., GradeBook).

Object Creation

The 3 steps of object declaration, creation and assignment

```
1          3   2
Dog myDog = new Dog();
```

1 Declare a reference variable

```
Dog myDog = new Dog();
```

Tells the JVM to allocate space for a reference variable, and names that variable *myDog*. The reference variable is, forever, of type Dog. In other words, a remote control that has buttons to control a Dog, but not a Cat or a Button or a Socket.

Dog

2 Create an object

```
Dog myDog = new Dog();
```

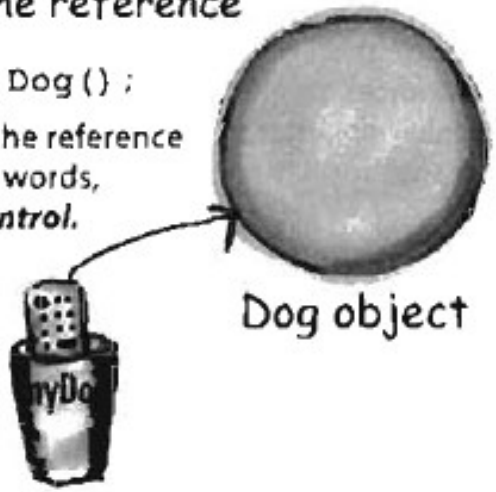
Tells the JVM to allocate space for a new Dog object on the heap (we'll learn a lot more about that process, especially in chapter 9.)


Dog object

3 Link the object and the reference

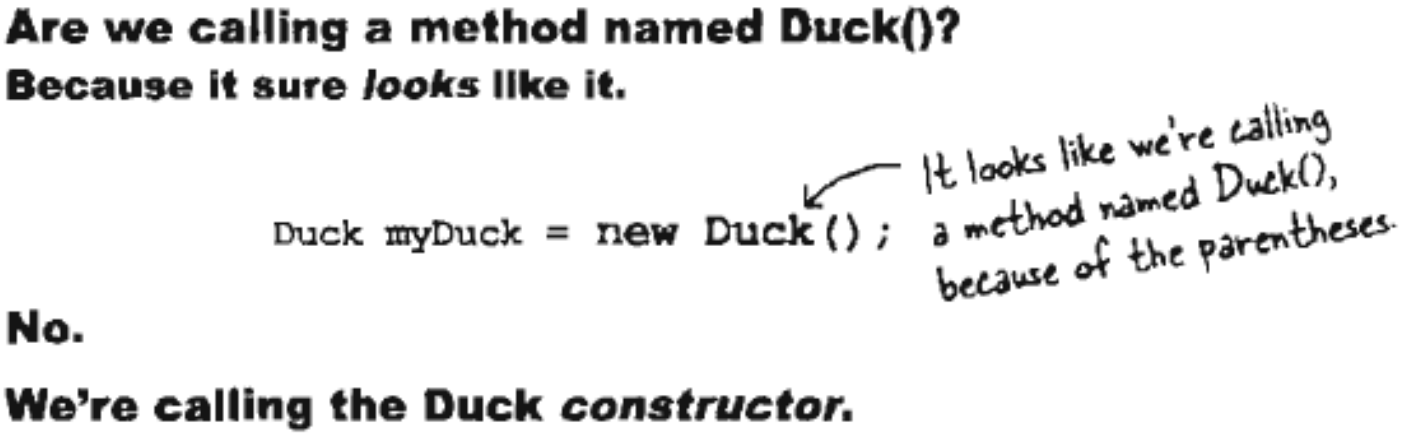
```
Dog myDog = new Dog();
```

Assigns the new Dog to the reference variable myDog. In other words, ***programs the remote control.***


Dog object

Dog

Constructors



A constructor *does look and feel a lot like a method, but it's not* a method. It has the code that runs when you instantiate an object.

The only way to invoke a constructor is with the keyword new followed by the class name.

But where is the constructor? In our previous programs, we didn't write It, who did?

You can write a constructor for your class, but if you don't, *the compiler writes one for you!*

Constructors

```
public class Duck {

    public Duck() {
        System.out.println("Quack");
    }
}
```

Constructor code.

```
public class UseADuck {

    public static void main (String[] args) {
        Duck d = new Duck();
    }
}
```

This calls the Duck constructor.

Object Creation: Using Constructors To Initialize Important Object States

constructor with arguments.

```
public class Duck {
   int size;

   public Duck(int duckSize) {
      System.out.println("Quack");

      size = duckSize;

      System.out.println("size is " + size);
   }
}
```

Add an int parameter to the Duck constructor.

Use the argument value to set the size instance variable.

```
public class UseADuck {

  public static void main (String[] args) {
     Duck d = new Duck(42);
  }
}
```

This time there's only one statement. We make the new Duck and set its size in one statement.

Pass a value to the constructor.

```
File Edit Window Help Honk
% java UseADuck
Quack
size is 42
```

*Not to imply that not all Duck state is not unimportant.

Constructors with arguments

```
class Box {
      double width; double height; double depth;
// This is the constructor for Box.
      Box(double w, double h, double d) {
      width = w; height = h; depth = d;
      }
                  // compute and return volume
                  double volume() {
                  return width * height * depth;
                  }
      }
 class BoxDemo3 {
      public static void main(String args[]) {
      // declare, allocate, and initialize Box objects
                  Box mybox1 = new Box(10, 20, 15);
                  Box mybox2 = new Box(3, 6, 9);
                  double vol;
      // get volume of first box
      vol = mybox1.volume();
      System.out.println("Volume is " + vol);
      // get volume of second box
      vol = mybox2.volume();
      System.out.println("Volume is " + vol);
      }
}
```

Note: Constructors

- It can be tedious to initialize all of the variables in a class each time an instance is created.
- Java allows objects to initialize themselves when they are created. This automatic initialization is performed through the use of a constructor.

- It is the constructor's job to initialize the internal state of an object so that the code creating an instance will have a fully initialized, usable object immediately.

3. If you don't put a constructor in your class, the compiler puts in a default *constructor.* The default constructor is always a no-arg constructor.

```
public Duck () { }
```

4. You can have more than one constructor in your class, as long as the argument lists are different. Having more than one constructor in a class means you have overloaded constructors.
    - public Duck () {
    - public Duck(int size) { }
    - public Duck(String name)
    - public Duck (String name, int size) { }

Encapsulation (Data Hiding), Getter and Setter Methods

- So far, we've been committing one of the worst OOP violation; that is Exposing our data (leaving our data out there for *anyone to see and even touch/change).*

- Exposed means reachable with the dot operator, as in:

dog.size = 30;

What if someone set value of our dog size equals to zero???

- We need to build setter methods for all the instance variables, and find a way to force other code to call the setter rather than access /manipulating instance variables directly.

Encapsulation (Data Hiding), Getter and Setter Methods

Encapsulation is a mechanism whereby data and codes are bound together as a single unit.

Through encapsulation, the methods and variables of a class are well hidden and safe.

Encapsulation in Java can be achieved by:

i. Declaring the variables of a class as private.
ii. Providing public setter and getter methods to modify and view the variables values.

Encapsulation (Data Hiding), Getter and Setter Methods

- We can hide our data by marking instance variables with private access modifier and provide public access modifier to setters and getters methods. Variables or methods declared with access modifier private are accessible ONLY to methods of the class in which they are declared.

- Encapsulation puts a force-field around instance variables, so nobody can set them to something inappropriate.

- By forcing other code to go through setter methods. That way, the setter method can validate the parameter and decide if it's do-able. Maybe the method will reject it and do nothing, or maybe the method will round the parameter sent into the nearest acceptable value.

Encapsulation (Data Hiding), Getter and Setter Methods

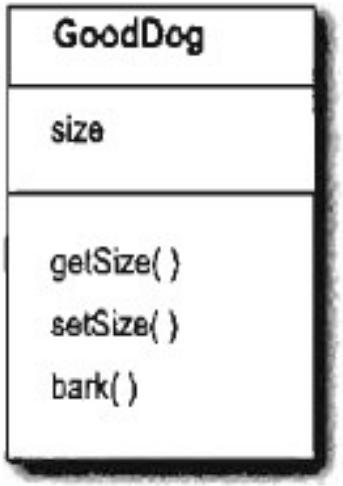
A method can be declared to give a specific type of value to the caller, such as:

```
int giveNumber () {
  return 42;
}
```

If you declare a method to return a value, you must return a value of the declared type (Or a value Compatible with the declared type).

Encapsulation (Data Hiding), Getter and Setter Methods

- Getter and setter methods facilitate to get and set instance variable values. Consider the following class diagram:



  - A Getter´s purpose is to send back, as a return value, the value of whatever it is that particular Getter is supposed to be getting.

  - A Setter´s purpose is to take an argument value and use it to set the value of an instance variable.

Getter and Setter Methods

```
Class Dog {
        private int size;

public void setSize(int s) {
        size = s;
}
public int getSize() {
        return size;
}



void bark (){
if (size < 1) {
        System.out.println (”Please  specify/enter a
        valid  size of a dog”);
} else if (size > 60) {
        System.out.println (”Woof ! Wooof!”);
} else if (size < 10) {
        System.out.println (”Yip! Yip!”);
} else  {
        System.out.println (”Ruff! Ruff!”);
                        }
        }
}
```

```
Class DogTest {

public  static void main (String  [] args) {

        Dog one = new Dog ();
        one.setSize (80);

        Dog two = new Dog ();
        two.setSize (5);

        System.out.println (”First Dog:” + one.getS
        System.out.println (”Second Dog:” +
        two.getSize());
        one.bark();
        two.bark();
        }
}
```